Journal of Statistical Software

January 2007, Volume 17, Issue 10. http://www.jstatsoft.org/

RAGE: A Java-implemented Visual Random Generator

Carlos Javier Pérez
Universidad de Extremadura

Hansgeorg Schwibbe
HAWK Hochschule für Angewandte Wissenschaft und Kunst

Petra Weidner
HAWK Hochschule für Angewandte Wissenschaft und Kunst

Abstract

Carefully designed Java applications turn out to be efficient and platform independent tools that can compete well with classical implementations of statistical software. The project presented here is an example underlining this statement for random variate generation. An end-user application called **RAGE** (Random Variate Generator) is developed to generate random variates from probability distributions. A Java class library called **JDiscreteLib** has been designed and implemented for the simulation of random variables from the most usual discrete distributions inside **RAGE**. For each distribution, specific and general algorithms are available for this purpose. **RAGE** can also be used as an interactive simulation tool for data and data summary visualization.

Keywords: Random numbers, discrete variates, statistical visualization, Java.

1. Introduction

The importance of simulation has increased in many areas of research and application during the past several years. Stochastic simulation is widely used to validate procedures and provide guidance for both theoretical and practical problems. Today, truly complex models often can only be computationally handled by simulation-based techniques.

Simulation methods require random variate generation. For this reason, many generation techniques have been developed. Some references on random variate generators are Devroye (1986), Ripley (1987) and Gentle (1998), among others. The current research trend is the

development of algorithms that are valid for large families of distributions. The programs based on these algorithms are called universal (also automatic or black-box) generators. In the last decade, some universal algorithms have been introduced, see Hörmann, Leydold, and Derflinger (2004) for a monograph on this topic.

The main focus of this project has been the design and Java-implementation of a program called **RAGE** (Random Variate Generator) to serve as an end-user application for generating random numbers. The Java class library **JDiscreteLib** has been developed for this project. **RAGE** uses **JDiscreteLib** to generate random variates from the most usual discrete distributions. For each distribution, specific and general algorithms to generate from are available.

RAGE can also be used as an interactive simulation tool for data and data summary visualization. Among the very important characteristics are:

- it is visual and dynamic.
- it is intuitive and user-friendly.
- it is programmed in Java, a robust and platform independent programming language.
- it can easily be extended for more functionality.
- the complete documentation is clearly presented in JavaDoc.
- it allows for saving the generated variates and their summaries as well as graphics.
- it is freely distributed under GNU General Public License.

The outline of the paper is as follows. Section 2 shows how **RAGE** can be installed, run and extended. The class library **JDiscreteLib** is described in Section 3. Finally, computational times for some examples are shown in Section 4.

2. The application RAGE

RAGE is an MDI (Multiple Document Interface) application where the content is implemented with plug-ins. This makes easy to extend the application with further functionality. The whole documentation is written in JavaDoc and can be generated from the source files by using the *javadoc* command. All source files are exported into the applications jar archives.

2.1. Installing RAGE

RAGE and the class library **JDiscreteLib** are published under the terms of the GNU General Public License and can be downloaded from `http://sourceforge.net/projects/rage`. Table 1 presents the files being available for the download. The program can be run on any platform that supports Java 1.5 or higher. The current version of the program is 1.06. Future updates will be placed under the same URL.

2.2. Running RAGE

This section describes how to run **RAGE** by presenting the main elements of its inferface. An important characteristic of the program is that its usage is easy and intuitive as it will be shown in this subsection.

Filename	Size (bytes)	Architecture
JDiscreteLib-V1.06.zip	649865	Platform-independent
RAGE-V1.06.zip	332667	Platform-independent

Table 1: Files for downloading on SourceForge.net.

Once the program has been installed on the computer and the `RAGE.jar` archive has been executed, an empty window appears on the screen. This window presents the general menus (*File*, *Distributions*, *Extras* and *Help*). The menu *Distributions* has been designed to allow the user to select the distribution he wants to generate random variates from. In the current program version, discrete distributions from the java class **JDiscreteLib** are available. These distributions are divided into distributions with finite support and distributions with infinite support as it is shown in Figure 1.

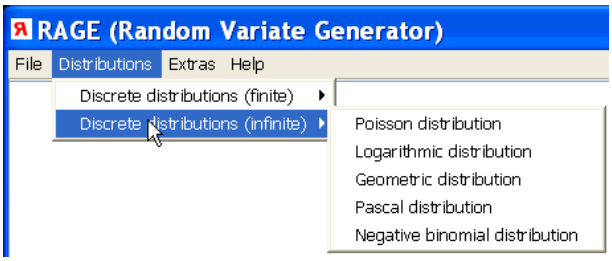

Figure 1: *Distributions* menu

After choosing the preferred distribution, a new window appears inside the main one. This window consists of four clearly differentiated parts as it is illustrated by Figure 2.

1. The first part (top-left) contains the default values for the parameters. They depend on the kind of distribution. The values can be changed by pushing the top/down arrows on the keyboard or by clicking on the top/down arrows inside the window. It is also possible to enter the parameter values by using the given text field.

2. The second one (top-right) presents the simulation inputs and options. The combo box *Algorithm* sets the algorithm to be used for the generation of the simulated values, whereas the combo box *Sample size* sets the number of random experiments to be carried out. The check box *Disable animation* allows to activate or deactivate the animation during the simulation. Animation enables a visualization of the goodness-of-fit through the time. Deactivating animation is useful to measure the computational times of the algorithms. Default values for sample size and animation can be changed in the main menu *Extras*. Finally, the button *Simulate* starts the simulation.

3. The third part (bottom-left) has been designed to show the main statistics and the probability distribution. The table at the top represents the statistics and the table at the bottom shows the probability distribution. For both tables, the blue colored values correspond to the exact distribution, and the red colored values stand for the simulated one.

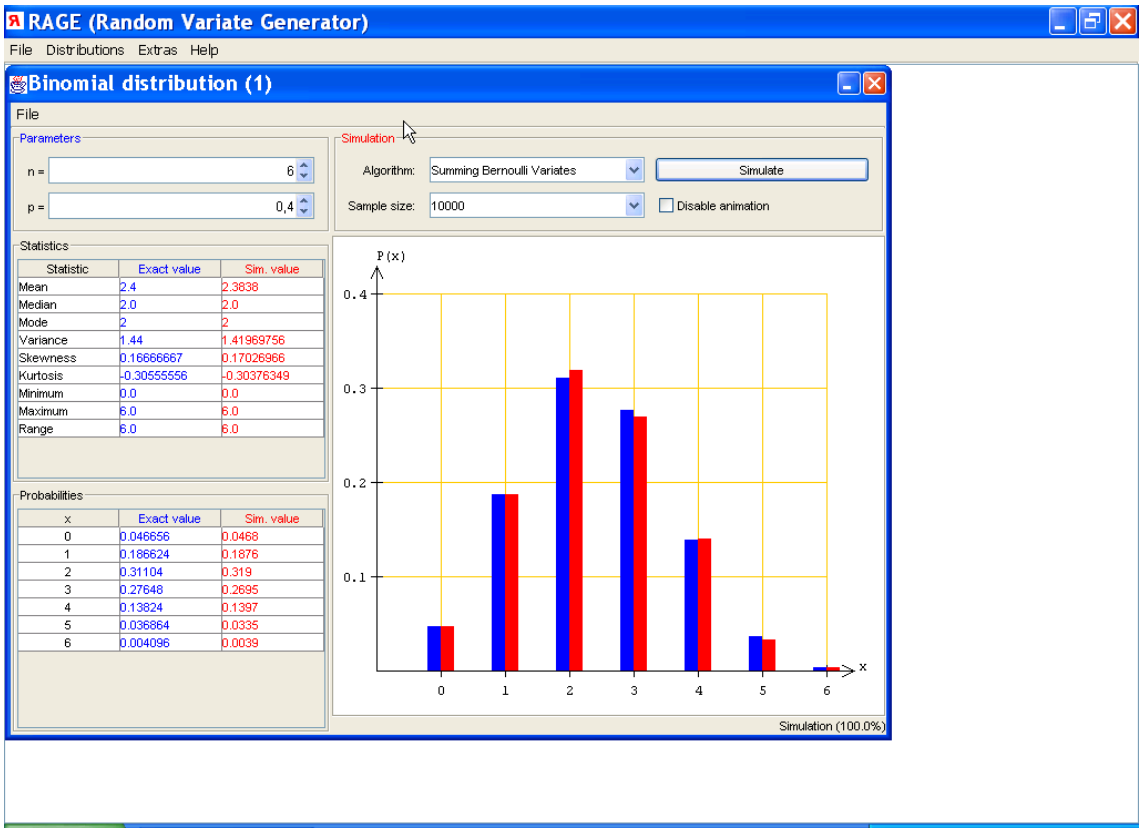

Figure 2: Window for a Binomial distribution

4. The last one (bottom-right) contains the graphics. The blue bars in the coordinate system show the exact probabilities for the selected distribution, while the red bars show the simulated values. The coordinate system can be rescaled by clicking on it with the mouse, allowing a better visualization of the barplot. The graphics for the infinite domain distributions are restricted by a threshold to show only the significant bars. In the *Extras* menu, the threshold can be selected to represent only the values with a probability larger than it. If the number of values for the variable is large and it does not allow a good visualization, then a "No graphics available" message is provided.

Finally, the menubar contains the *File* menu that provides some loading and saving options. The menu item *Load XML* opens a dialog window where the user has the possibility to load a previously saved XML (eXtended Markup Language) file. *Save data* opens a dialog window where the user is able to save the generated data and summary data as an XML or ASCII file. The XML file contains all the necessary information to rebuild the window (parameters, true and simulated values, graphics,...). The menu *Save graphics* opens a dialog window that allows the user to save the graphics as a PNG (Portable Network Graphics) or JPEG (Joint Photographic Expert Group) file.

2.3. Extending RAGE

The Java class for the generation of random numbers included in **RAGE** is **JDiscreteLib** (see Section 3). This class contains algorithms to generate random numbers from univariate discrete distributions. New classes can be added, for example for the generation of random numbers from univariate continuous distributions.

The application has been designed by using plug-ins in order to ease future extensions that provide more functionality. New plug-ins can be included in the following way:

1. Create a new Java project.
2. Bound the jar archive `RAGE.jar` into the project.
3. Create a new plug-in class which is derived from the class "`mdiframework.MDIPlugin`". For example:

```
import java.awt.BorderLayout;
import java.util.Locale;
import javax.swing.;
import mdiframework.MDIPlugin;

public class MyPlugin extends MDIPlugin {
  private JInternalFrame iFrame = null;
  private JLabel jLabel = null;
  public JInternalFrame getJInternalFrame() {
    if (this.iFrame == null) {
      this.iFrame = new JInternalFrame();
      this.iFrame.getContentPane()
        .setLayout(new BorderLayout());
      this.iFrame.getContentPane()
        .add(this.getJLabel(),
          BorderLayout.CENTER);
      this.iFrame.setSize(400, 400);
    }
    return this.iFrame;
  }
  public JLabel getJLabel() {
    if (this.jLabel == null) {
      this.jLabel = new JLabel();
      this.jLabel.setText("Hello plug-in !!!");
      this.jLabel.setHorizontalAlignment(
        SwingConstants.CENTER);
    }
    return this.jLabel;
  }
  public String getPluginName() {
    return "My plug-in";
  }
```

```
  public int getPreferredIndex() {
    return 1;
  }
  public String getSubmenuName() {
    return "New extensions";
  }
  public void setResourceBundle(Locale locale) {
    // Optional: Set the resource bundle:
    /* this.bundle = ResourceBundle
        .getBundle("PluginBundle", locale); */
  }
}
```

4. Export the project as a jar archive.
5. Copy the exported jar archive into the directory "plug-ins" inside the **RAGE** folder.
6. Start **RAGE** with the new plug-in.

Future extensions of **RAGE** will include new distributions with the corresponding generating algorithms. The direct extension is for univariate continuous distributions. Specific and general generating algorithms will be included in a future version. Also, both discrete and continuous multivariate distributions will be considered. It is remarkable that in multidimensional settings everything becomes more difficult. However, an important number of general and particular algorithms have been designed for multivariate distributions (see, for example, Hörmann *et al.* (2004)). In this case, the multidimensional extension should include modifications in the **RAGE** program to accommodate the new situation, e.g., to show the graphics and statistics in different windows for each vector component. Finally, tools to compare algorithms in terms of efficiency will be implemented.

The complete information about the program structure and the functionality can be found in the documentation included in JavaDoc.

3. The class library JDiscreteLib

The class library **JDiscreteLib** contains random generators for the most usual univariate discrete distributions with finite as well as with infinite support (Uniform, Bernoulli, Binomial, Hypergeometric, User-supplied, Poisson, Logarithmic, Geometric, Pascal and Negative Binomial). Specific and general algorithms have been implemented.

Specific algorithms inside **JDiscreteLib**:

- the inverse transformation method (*Basic uniform*) for the uniform distribution,
- the generator for the Bernoulli distribution (*Basic Bernoulli*) according to Gentle (1998), page 47,
- the *Summing Bernoulli Variates* algorithm for the Binomial distribution (Ripley 1987, page 78),

- the *Geometric Method* for the Binomial distribution as in Devroye (1980),
- the *Uniform Variate Multiplication* algorithm (Devroye 1986, page 504) for the Poisson distribution,
- the *Exponential Interarrival Times* method (Bratley, Fox, and Schrage 1987, page 182) for the Poisson distribution,
- an algorithm based on a *Distributional Property* (Devroye 1986, page 547) for the logarithmic distribution,
- the *Exponential Variates* method (Ripley 1987, page 77) for the Geometric distribution,
- the *Experimental Method* (Devroye 1986, page 499) for the Geometric distribution,
- the *Summing Geometric Variates* algorithm (Devroye 1986, page 543) for the Pascal distribution,
- the *Composition Method* (Gentle 1998, page 101) for the Negative Binomial distribution.

Moreover, for distributions with finite support, the Alias method, inversion by sequential search and inversion by indexed search are implemented, whereas for distributions with infinite support, an indexed search tail method has been implemented. All the algorithms derived from these methods are general and not specific for any kind of discrete distributions. The *Alias Method* (Walker (1977)) is an algorithm that uses aliases and bar fractions. The bar fractions and the aliases have to be calculated once at the beginning before generating random variates. Inversion by sequential search is implemented as a slightly modified version of the algorithm given in Hörmann *et al.* (2004), page 43, to optimize speed and performance by using cumulative probabilities. In this version, the cumulative probability vector is calculated during the setup and stores the result in memory instead of having to evaluate the sum of probabilities each time a value is generated. This makes the algorithm much faster when the sample size is moderate or large. *Indexed Search* and *Indexed Search Tail* are implemented as described on page 45 and page 216 of Hörmann *et al.* (2004) by including the cumulative probabilities in the setup step as in the previous algorithm.

4. Computation time comparisons

Carefully designed Java applications run fast enough on today's hardware and can compete well with classical implementations. This section shows how the proper selection of an algorithm can save an important amount of computational time, allowing an efficient random variate generation. Of course, the computational time depends on a variety of factors as the computer used, the algorithms, the quality of the implementation, the kind of distribution to generate from or the parameters. Speed is one criterion for the evaluation of random variate generators, but not the only one. A tradeoff between computational time, implementation effort and memory requirements has to be considered. A general choice is to use universal or automatic algorithms that are valid for large families of distributions as, for example, those included in Hörmann *et al.* (2004) and Marsaglia, Tsang, and Wang (2004).

In this section, time comparisons are presented for both a discrete distribution with finite support and a discrete distribution with infinite support by using **RAGE**. The graphical

animation must be disabled to show the execution time for the generating process (not for calculating statistics or plotting the final graphics). The computational time is presented in the bottom-right part of the window. The machine used is a Pentium IV PC operating at 3.0 GHz.

One million random variates from Binomial distributions with several parameters have been generated. Table 2 shows the execution time in seconds for several methods. The methods used are: Summing Bernoulli Variates (SBV), Geometric Method (GM), Sequential Search (SS), Indexed Search (IS), and Alias Method (AM).

n	p	SBV	GM	SS	IS	AM
5	0.1	1.172	1.063	0.313	0.312	0.500
	0.4	1.203	1.875	0.297	0.328	0.500
10	0.1	2.297	1.344	0.313	0.328	0.515
	0.4	2.360	2.937	0.328	0.329	0.515
25	0.1	5.610	2.125	0.375	0.328	0.500
	0.4	5.813	6.125	0.391	0.343	0.532
50	0.1	11.234	3.469	0.484	0.328	0.515
	0.4	11.562	11.359	0.515	0.391	0.562
75	0.1	16.735	4.781	0.594	0.343	0.515
	0.4	17.328	16.593	0.640	0.407	0.579
100	0.1	22.375	6.078	0.687	0.359	0.516
	0.4	23.079	21.985	0.766	0.454	0.625

Table 2: Computational times in seconds for Binomial variates.

Similarly, one million random variates from Poisson distributions with several parameters have been generated. Table 3 show the execution time in seconds for several methods. The methods used are: Uniform Variate Multiplication (UVM), Exponential Interarrival Times (EIT), and Indexed Search Tail (IST). Both Table 2 and Table 3 shows that very important computational differences are obtained from one method to another.

Acknowledgements

Suggestions from Jacinto Martín are gratefully acknowledged. This work has been partially supported by *Ministerio de Educación y Ciencia*, Spain (Project TSI2004-06801-C04-03).

References

Bratley P, Fox BL, Schrage LE (1987). *A Guide to Simulation*. Springer-Verlag.

Devroye L (1980). "Generating the Maximum of Independent Identically Distributed Random Variates." *Computer and mathematics with applications*, **6**, 305–315.

Devroye L (1986). *Non-Uniform Random Variate Generation*. Springer-Verlag.

λ	UVM	EIT	IST
0.1	0.937	0.485	0.328
0.2	0.953	0.516	0.328
0.3	0.984	0.546	0.328
0.4	1.063	0.625	0.328
0.5	1.078	0.656	0.328
0.6	1.094	0.688	0.328
0.7	1.125	0.719	0.328
0.8	1.175	0.734	0.328
0.9	1.266	0.766	0.328
1	0.735	0.782	0.328
5	2.250	1.735	0.359
10	3.485	2.890	0.360
25	7.141	6.375	0.406
50	13.203	12.140	0.438
75	19.203	17.922	0.454
100	25.313	23.891	0.485

Table 3: Computational times in seconds for Poisson variates.

Gentle JE (1998). *Random Number Generation and Monte Carlo Methods*. Statistics and Computing. Springer-Verlag.

Hörmann W, Leydold J, Derflinger G (2004). *Automatic Nonuniform Random Variate Generation*. Statistics and Computing. Springer-Verlag.

Marsaglia G, Tsang WW, Wang J (2004). "Fast Generation of Discrete Random Variates." *Journal of Statistical Software*, **11**(3), 1–11.

Ripley BD (1987). *Stochastic Simulation*. Wiley Series in Probability and Mathematical Statistics. John Wiley and Sons.

Walker AJ (1977). "An Efficient Method for Generating Discrete Random Variables with General Distributions." *ACM Transaction on Mathematical Software*, **3**, 253–256.

Affiliation:

Carlos Javier Pérez
Departamento de Matemáticas
Facultad de Veterinaria
Universidad de Extremadura
Avda. de la Universidad s/n
10071 Cáceres, Spain
Fax: +34/927257110
E-mail: carper@unex.es

Hansgeorg Schwibbe
Fakultät Naturwissenschaften und Technik
HAWK Hochschule für Angewandte Wissenschaft und Kunst
Von-Ossietzky-Str. 99
D-37085 Göttingen, Germany
E-mail: hansgeorg.schwibbe@gmx.de

Petra Weidner
Fakultät Naturwissenschaften und Technik
HAWK Hochschule für Angewandte Wissenschaft und Kunst
Von-Ossietzky-Str. 99
D-37085 Göttingen, Germany
E-mail: weidner@hawk-hhg.de

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 17, Issue 10 *Submitted:* 2006-03-30
January 2007 *Accepted:* 2007-01-31